Definition of an eXecutable SPEM 2.0

Réda Bendraou[†] Benoît Combemale[‡] Xavier Crégut[‡] Marie-Pierre Gervais[†]

[†] *Laboratoire d'Informatique de Paris 6* (CNRS UMR 7606), Paris, France
first_name.last_name@lip6.fr
[‡] *Institut de Recherche en Informatique de Toulouse* (CNRS UMR 5505), Toulouse, France
first_name.last_name@enseeiht.fr

Abstract

One major advantage of executable models is that once constructed, they can be run, checked, validated and improved in short incremental and iterative cycles. In the field of Software Process Modeling, process models have not yet reached the level of precision that would allow their execution. Recently the OMG issued a new revision of its standard for Software Process Modeling, namely SPEM2.0. However, even if executability was defined as a mandatory requirement in the RFP (Request For Proposal), the adopted specification does not fulfill it. This paper presents a critical analysis on the newly defined standard and addresses its lacks in terms of executability. An approach is proposed in order to extend the standard with a set of concepts and behavioural semantics that would allow SPEM2.0 process models to be checked through a mapping to Petri nets and monitored through a transformation into BPEL.

1 Introduction

Since the earliest projects developing large software systems, one main concern of organizations was to provide a conceptual scheme for rationally managing the complexity of software development activities. Several Software Development Life Cycle (SDLC) have been proposed like the "Waterfall", "Spiral" and the "Incremental Model". However, the software process modeling community was unsatisfied with using these life-cycle descriptions. The granularity of SDLC models is too coarse-grained and fails to describe elementary process building blocks [7]. Rapidly, the need to describe in more details processes that companies are actually performing during software development or maintenance emerged. The idea was to decompose these SDLC descriptions into sufficient detail so that they can provide more explicit guidance for executing a software development project. This is how the notion of *Process Models* (PM) appeared.

Goals that motivated the introduction of process models are manifold: they span from informal support and facilitating human understanding to direct assistance in process assessment, management and enactment [22]. To be most effective in supporting this variety of objectives, process models must go beyond representation. The understanding of process participants about the contents and sequencing of process steps depends strongly on the degree of details provided in the process model. Recently, the pressure for greater granularity (i.e., more details) in process models is driven by the need to ensure process precision, the degree to which a defined process specifies all process steps needed to produce accurate results [9]. Another pressure comes from the increasing demand for process validation and automation, which requires precise process models at relatively deep levels of detail.

In this paper, we will focus on two main objectives. The first one deals with the ability to validate a software process model at any stage of its lifecycle, i.e., during its design, when it is tailored for a given project and while it is conducted. Validation may be achieved through the behavioral checking of process model properties. The second objective relates to the automation of the execution support. We mean by execution support, the ability to monitor and to control a real process according to its defined process model. We try to reach these two objectives i.e. *Validation* and *Execution* in the context of SPEM2.0 (*Software Process Engineering Metamodel*) [18], the recently adopted OMG's standard dedicated to software process modeling. Addionnally to describing a concrete software development process or a family of related software development processes, the last version of the standard claims its ability for partially supporting process enactment, a facility which was intentionally kept out of the scope in SPEM1.1 [15].

[0]This work is supported by the IST European project MODELPLEX (contract n° IST-3408) and the TOPCASED project.

In the next section of this paper, we will start by giving a critical analysis on the enactment approaches proposed in SPEM2.0 and we will address their limits in supporting process model executions. In order to tackle these limits, in section 3, we define an extension of SPEM2.0 that allows the specification of executable process models. This extension brings a new compliance level to the standard. It comes in form of a set of elements and features that merge the SPEM2.0 metamodel. We also associate a behavioral semantics to the proposed extension. For validating and ensuring that processes can be executed with respect to constraints defined in the process model, an approach is proposed in section 4. Examples of such constraints are schedulability constraints, temporal constraints and those related to resource availabilities. The approach uses model-checking. At this aim, process models are translated into a formal language: Petri nets. This has the primary advantage of leveraging the myriad of already existing model-checkers. In section 5, we explore the possibility to use workflow tools for monitoring purposes. Thus, we propose a mapping between SPEM elements and BPEL (*Business Process Execution Language*) concepts. Finally, section 6 concludes this work.

2 Process Models Enactment with SPEM2.0

SPEM2.0 is the OMG's standard dedicated to software process modeling. It aims at providing organizations with means to define a conceptual framework offering the necessary concepts for modeling, interchanging, documenting, managing and presenting their development methods and processes [18]. Besides providing a standard way for representing organization's processes and expertise, SPEM2.0 comes with a new attractive vision. That latter consists in separating all the aspects, contents and material related to a software development methodology from their possible instantiation in a particular process. Thus, to fully exploit this framework, the first step would be to define all the phases, activities, artifacts, roles, guidance, tools, and so on, that may compose a methodology and then, to pick, according to the situation or process context, the appropriate method contents to use within a process definition.

SPEM2.0 comes in form of a MOF-compliant metamodel [17] that reuses UML2.0 Infrastructure [19] and UML2.0 Diagram Interchange specifications [16]. It reuses from the UML Infrastructure basic concepts such as *Classifier* or *Package*. No concept from the UML2.0 Superstructure [20] is reused. The Standard comes also in form of UML Profile where each element from the SPEM2.0 metamodel is defined as a stereotype in UML2.0 Superstructure. The metamodel is composed of seven packages linked with the "merge" mechanism (cf [19], §11.9.3), each package dealing with a specific aspect (cf. Fig. 1). The *Core*

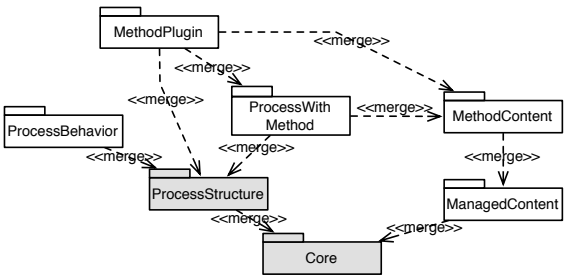

Figure 1. Structure of SPEM2.0 [18]

package introduces classes and abstractions that build the foundation for all other metamodel packages. The building block of this package is the *WorkDefinition* class, which generalizes any work within SPEM2.0. The *Process Structure* package defines elements for representing basic process models in terms of a flow of *Activities* with their *WorkProduct Uses* and *Roles Uses*. However, the possibility to textually document these elements (i.e., add properties describing the element) is not provided in this package but in the *Managed Content* package, which provides concepts for managing the textual description of process elements. Examples of such concepts are the *Content Description* class and the *Guidance* class. The *Method Content* package defines core concepts for specifying basic method contents such as *Roles*, *Tasks* and *WorkProducts*. The *Process with Method* package defines the set of elements required for integrating processes defined by means of *Process Structure* package concepts with instances of *Method Content* package concepts. The *Method Plugin* package provides mechanisms for managing and reusing libraries of method contents and processes. This is ensured thanks to the *Method Plugin* and *Method Library* concepts. Finally, *Process Behavior* package provides a way to link SPEM2.0 process elements with external behavior models such as UML2.0 Activity Diagrams or BPMN (*Business Process Modeling Notation*) models.

However, even if process enactment was among the principal requirements when the SPEM2.0 RFP was issued [14], the recently adopted specification does not address the enactment issue. Nevertheless, it clearly suggests two possible ways of enacting SPEM2.0 process models. In the following, we introduce them, we present the concepts that are supposed to be used in order to enact SPEM2.0 models and, we give some remarks on the feasibility of each approach.

2.1 Mapping the SPEM2.0 Processes Models into Project Plans

In this first approach the standard proposes to map SPEM2.0 processes into project plans by means of project planning and enactment systems such as IBM Rational Portfolio Manager or Microsoft Project. Once SPEM2.0 pro-

cesses are mapped to project plans, the plans can be instantiated by means of planning tools and concrete resources can then be affected. However, whether this approach may be very useful for project planning, it is not considered as process enactment. It is still necessary to affect duration to tasks, persons to roles in order to get, at the end, an estimation of the development process period and resources needed for its realization. These plans are used by a project manager in order to estimate if the process will be in schedule or not, whether more persons need to be affected to process tasks, etc. There is no support for process execution, no automatic task affectations to responsible roles, no automatic routing of artifacts, no automatic control of work product states after each activity, no means to support agent and team communication and so on. Besides the fact that this approach does not provide concrete enactment support, it presents a major lack which is its tight dependence to the project planning tool. Another aspect that has to be taken into account is the impact of modifying or adding information within the planning tool and how this modification will be reflected / traced-up to the SPEM2.0 process model. Finally, process modelers have to deal with the compatibility of the process definition file format of the planning tool.

2.2 Linking SPEM2.0 process elements with external behavior formalisms

The SPEM2.0 standard does not provide any concepts or formalism for modeling precise process behavior models or execution. Rather, claiming for more flexibility, SPEM2.0 provides, through the *Process Behavior* package, a way to link SPEM2.0 process elements with external behavior models. The goal behind is not to restrict or to impose a specific behavior model but to give the process modeler the option to choose the one that fits his needs best. Further more, a mapping towards BPEL can be carried out in order to reuse BPEL execution engines. In addition, a *WorkProduct* can for instance be linked to a UML state diagram in order to model possible *WorkProduct*'s states and transitions that can make this *WorkProduct* move from one state into another. Here again, a state machine engine has to be integrated to the process execution engine. SPEM2.0 defines a kind of proxy classes (i.e., *Activity_ext*, *ControlFlow_ext*, *Transition_ext* and *State_ext*) in order to link between SPEM2.0 process elements (i.e., *WorkProductUse*, *WorkDefinition*, *RoleUse*, *Activity*, *WorkSequence*) and external behavior model elements. It is up to the process modeler to link each process element with its equivalent in the behavior model. Since a single behavior model may not be expressive enough to represent all the behavioral aspects of the process, several behavior models can be combined.

Even if this approach may provide flexibility in representing behavioral aspects of SPEM2.0 processes, it presents some lacks. The first one is that the standard is not very clear on how the linking of process elements with behavioral models is realized. It just provides proxy classes that make reference to other elements in an external behavioral model. We suppose that this task is tool implementer's responsibility. A tool implementers have to define a specific behavioral model that has to be automatically generated from the SPEM2.0 process model. This is already the case in the free EPF[1] tool which is meant to be the implementation of SPEM2.0. In EPF, a kind of a proprietary activity diagram is partially generated from a process definition. The latter can be refined in order to provide more details on the process activities and their coordination (control flows). However no execution is provided. The second lack is that the mapping from SPEM2.0 process elements into a specific behavioral model can be done differently from one organization to another, depending on the process modeler's interpretation. Thus, a standardization effort may be required in order to harmonize mapping rules between SPEM2.0 concepts and a specific behavior model such as BPEL for instance. At this aim, in section 5, we propose such mapping rules between a subset of SPEM2.0 concepts and the BPEL language. The third lack, which tightly relates to the previous one, is that more often concepts in behavior models are richer than in SPEM2.0. This is due to the fact that behavior modeling and execution languages provide additional concepts related to the technical support and execution of processes while SPEM2.0 concentrates on the "business concerns" of the software development process or methodology (i.e., *Roles*, *Activities*, *Guidance*, etc.). Consequently, a full executable code generation from SPEM2.0 is not possible which may impose some refinement steps in behavior models before they can be enacted. This in its turn poses the problem of traceability and how these refinements (changes) can be reflected in the initial SPEM2.0 model.

In the rest of the paper, we propose an approach for validating and executing SPEM2.0 process models. For this purpose, we need first to extend the SPEM2.0 metamodel in order to take into account some process execution aspects.

3 xSPEM: SPEM2.0 Extension for Process Enactment

In this section, we propose to extend the SPEM2.0 specification in order to take into account the support of process enactments while remaining standard. We call our proposition, xSPEM which stands for *eXecutable SPEM*. For sake of clarity, we only present the minimal subset of SPEM2.0 concepts required for process execution. These concepts are regrouped in the xSPEM *Core* package (section 3.1). Additional features are required in the purpose of tailor-

[1] *Eclipse Process Framework* Project, `http://www.eclipse.org/epf`

ing a process for a given project. This includes defining properties specific to activity scheduling and resource allocations. They are introduced in xSPEM *ProjectCharacteristics* package (section 3.2). During execution, the process will evolve from one state into another. It is then required to provide concepts for characterizing all process states during enactment time. This is the aim of the xSPEM *ProcessObservability* package (section 3.3). We also define the behavioral semantics needed for process execution. Finally, we defined events that trigger state changes in the xSPEM *EventDescriptions* package (section 3.4).

3.1 xSPEM Core

xSPEM *Core* reuses concepts offered by the *Core* and the *ProcessStructure* packages (grey packages on fig. 1). Fig. 2 shows xSPEM Core's concepts. An *Activity* is a concrete *WorkDefinition* that represents a general unit of work assignable to specific performers represented by *RoleUse*. It can rely on inputs and produces outputs represented by *WorkProductUses*. *RoleUse* models a set of competences required to perform an activity (*ProcessPerformerMap*). An activity may be broken down into sub-activities. Activities are ordered thanks to the *WorkSequence* concept whose attribute *linkKind* indicates when an activity can be started or finished. We do not reuse the remaining packages as their content do not have any impact on process executability.

3.2 xSPEM Project Characteristics

In order to tailor a process model for a given project, additional features have to be defined. It is required to dimension activities, i.e., specify the number of used resources, expected duration, etc., and to identify the concrete resources allocated to the project. In this paper, we consider that *RoleUse* and *WorkProductUse* may be considered as resources required to perform an activity. The *direction* attributes defined in *WorkDefinitionParameter* could be used to complete sequencing constraints expressed through the *WorkSequence* concept. They are not detailed in this paper due to lack of space but they would be treated in the same way as activities described below.

We have thus defined the *xSPEM_ProjectCharacteristics* package as an extension of the *xSPEM_Core* package using the OMG *merge* operator. It redefines the concepts of *Activity*, *RoleUse* and *ProcessPerformerMap* by adding: 1) the time interval during which an activity must finish (*min_time* and *max_time* on *Activity*); 2) the number of role occurrences required to perform an activity (*occurrencesNb* on *RoleUse*); 3) the work load affected to a role for an activity (*charge* on *ProcessPerformerMap*).

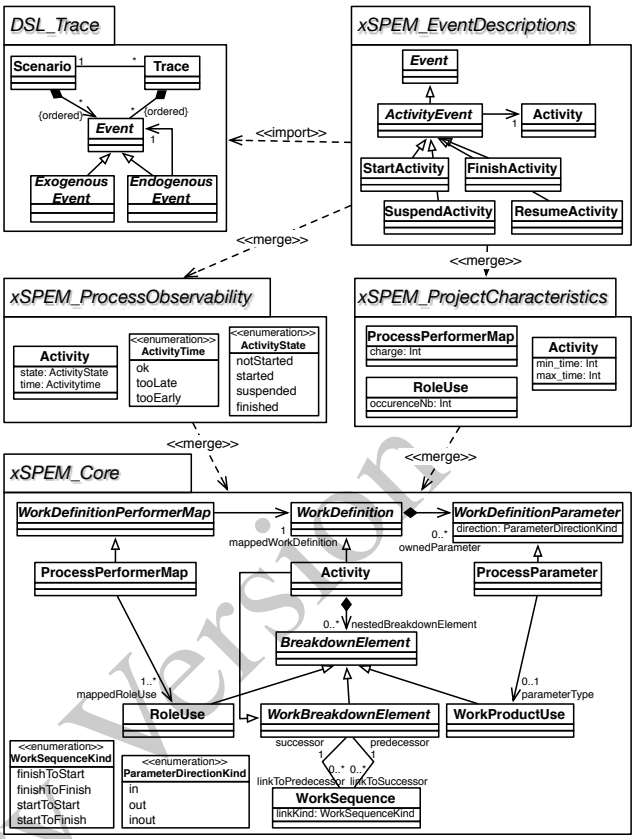

Figure 2. xSPEM**: an eXecutable SPEM2.0**

3.3 xSPEM Process Observability

In order to enact a process model, its semantics has to be defined. We apply a property-driven approach that we have described in [5]. It helps a Domain Specific Language (DSL) expert in defining the DSL semantics. It first consists in identifying relevant properties and specifying them using a temporal logic. Then, they are used to point out states of interest for the DSL expert and transitions that lead from one state to another.

The formal semantics associated to the system can be seen as the set of maximal finite traces whose elements are model states. If the metamodel has a well defined operational semantics, it can be easily expressed as a modification of instance's attributes or a modification of the topology (dynamically creating or deleting instances). On the contrary, if the associated semantics is not formally defined, the states characterised by properties allow to define an observable operational semantics. Following this idea, if state properties rely on notions that cannot be directly expressed in the model (classical OCL queries), then the metamodel must be enriched to express these notions. The dynamic operational semantics, i.e. the Kripke structure that allows to build trace semantics, must then be approximated by defining transitions between characterised states. It is the work of the domain expert to describe them.

This approach has three major advantages: it gives a method to define a formal semantics, it is incremental (properties may be added one after another) and it allows to easily define an observable approximation of the trace semantics.

Based on expertise described in [1], we now apply our property-driven approach to define the behavioral semantics of SPEM2.0 metamodel.

3.3.1 Characterising Properties

We identify the following properties that every xSPEM model must satisfy. We split them in two classes; *universal properties* that have to be satisfied by every execution (every activity must start, all started activities must finish, all suspended activities must resume, once an activity is finished, it has to stay in this state, an activity is able to start or finish depending on worksequences constraints...) and *existential properties* that must be true in at least for one execution (each activity must be performed in more than *min_time* and less than *max_time*, the overall process can to finish when all activities are finished between *min_time* and *max_time*).

Similar properties are defined on products. For example, an activity can only be started if its input products have been started (at least one activity producing them is started) or completed (all the activities producing them are finished). It could be of interest to know whether it is possible to complete all the products of a process.

3.3.2 Extending the Metamodel to Represent Dynamic Informations

The second step consists in adding features to the metamodel to capture states implied by the aforementioned properties. For xSPEM, we can identify two orthogonal aspects for the *Activity* element. First, an activity can be *not started*, *started*, *suspended* and finally *finished*. Secondly, there is a notion of time and clock associated to each activity; but this time is only relevant for transition-enabling conditions (in our case transitions that start and finish an activity) and is not explicit in state properties. Thus it can be represented into the finite set of states $\{tooEarly, ok, tooLate\}$. This second orthogonal aspect is only relevant when the activity is finished.

Now we have to express these states by extending the *Activity* element in order to introduce attributes that reflect dynamic informations, i.e. the state of the current activity. We choose to add two attributes (in the *xSPEM_ProcessObservability* package): $state \in \{notStarted, started, suspended, finished\}$ and $time \in \{tooEarly, ok, tooLate\}$. It is also necessary to take into account the concept of a clock ($clock \in \mathbb{R}^+$), internal to an activity. It is not represented in the metamodel because only the abstraction is necessary, the clock being taken into account by the execution engine.

An observational abstraction of the operational semantics of our processes with respect to our properties can now be defined. The expert has again to formalise the initial state and the transition relation. In our case, it is quite natural: the initial state is $\{a \mapsto (notStarted, ok) | a \in \mathcal{A}\}$. The transition relation is defined for *Activity* in Fig. 3.

3.4 xSPEM Events Description

Definition of the observational semantics introduces states and transitions. States have been captured through features added on the xSPEM metamodel. Transitions are triggered by events that make the process evolve. Events can be exogenous (produced by the environment of the process) or endogenous (produced within the process). For example the transition between the states *not started* and *started* corresponds to the event *StartActivity* applied on the corresponding activity. Other events on an *Activity* are *finish*, *suspend* and *resume*. They are modelled in the *xSPEM_EventsDescription* package as specializations of *ActivityEvent*, an abstract event that involves a target activity.

Events may be recorded as a trace so as to keep track of what happened in the process (package *DSL_Trace*). They can also be used to build scenarios that can be used to simulate a process.

4 Process Models Validation

In this section, we propose to implement semantics defined in the previous section in order to check xSPEM process models. Like for programming languages, there are several approaches.

Operational semantics allows to precisely describe the dynamic behavior of the language's constructions. In MDE (Model-Driven Engineering), it aims to express the behavioral semantics of a metamodel and thus provides executable models. We explore in [6] two ways to achieve this purpose. The first one is closer to the operational semantics in programming languages (Structural Operational Semantics [21], natural semantics [12]). It consists in defining transformations between two execution states of a model (for example, in the ATL [11] transformation language). The whole set of transformations defines the behavior of models. The second way consists in describing the behavior of each concept of the metamodel in an imperative manner using metaprogramming languages such as Kermeta [13], xOCL [4] or an action language.

Translational semantics relies on a well-defined formalism to express the semantics of a given language [8]. A translation is carried out from all concepts of the source language towards this formalism. This translation defines

Let a be the considered activity.

$$\forall ws = a.predecessor, (ws.linkType = startToStart \&\& ws.linkToPredecessor.state = started) \\ ||(ws.linkType = finishedToStart \&\& ws.linkToPredecessor.state = finished)$$

$$notStarted, ok, clock \xrightarrow{StartActivity} started, ok, 0$$

$$started, ok, clock \xrightarrow{SuspendActivity} suspended, ok, clock$$

$$suspended, ok, clock \xrightarrow{ResumeActivity} started, ok, clock$$

$$\forall ws = a.predecessor, (ws.linkType = startToFinished \&\& ws.linkToPredecessor.state = started) \\ ||(ws.linkType = finishedToFinished \&\& ws.linkToPredecessor.state = finished)$$

$$started, ok, clock < min\_time \xrightarrow{FinishActivity} finished, tooEarly, clock$$

$$started, ok, clock \in [min\_time, max\_time] \xrightarrow{FinishActivity} finished, ok, clock$$

$$started, ok, clock > max\_time \xrightarrow{FinishActivity} finished, tooLate, clock$$

Figure 3. Event-based Transition Relation for Activities

the semantics of the source language. In MDE, it consists in translating towards another technical space [4]. This approach allows to take advantage of all tools available in the target domain. This last approach is explored in the section hereunder in order to re-use model-checkers available in the model-checking community.

4.1 Translational Semantics to Petri Nets

In this experimentation, we choose to use the technical space of timed Petri nets as the target representation for formally expressing XSPEM process models. We also choose to generate our properties as LTL (*Linear Temporal Logic*) formulae over the Petri net associated to a process model. Then we manipulate timed Petri nets and LTL formulae within the *Tina*[2] toolkit [2] which includes: 1) *nd* (*NetDraw*), an editing tool for automata and timed networks, under a textual or graphical form. It integrates a "step by step" simulator (graphical or textual) for the timed networks. 2) *Tina*, which builds the state space of a Petri net, timed or not. *Tina* performs classical constructs (marking graphs, covering trees) and allows abstract state space construction, based on partial order techniques. 3) *selt*, a *model-checker* for formulae of an extension of temporal logic *seltl* (State/Event LTL) [3]. In case of non satisfiability, *selt* is able to build a readable counter-example sequence usable by the *TINA* simulator to execute it step by step.

The XSPEM semantics is defined as a mapping to Petri nets (XSPEM2PETRINET). A *PetriNet* is composed of *Nodes* that denote *Places* or *Transitions*. Nodes are linked by *Arcs*. Arcs can be normal ones or read-arcs. An *Arc* specifies the number of tokens consumed in the source place or produced in the target one. A *read-arc* only checks tokens availability without removing them. Petri nets marking is defined by the number of tokens contained in places. Finally, a time interval can be expressed on *Transitions*.

An example of transformation from a process model to a Petri net model is given in Fig. 4. Each activity is translated into five places characterising its state (*NotStarted*, *Started*, *InProgress*, *Suspended* or *Finished*). The state *Started* records that the activity has been started. A *WorkSequence* becomes a *read-arc* from one place of the source activity to a transition of the target activity. We also add five places that define a local clock. The clock will be in state *TooEarly* when the activity ends before *min_time* and in the state *TooLate* when the activity ends after *max_time*. This transformation has been written in ATL within the context of an execution dedicated XSPEM subset[3].

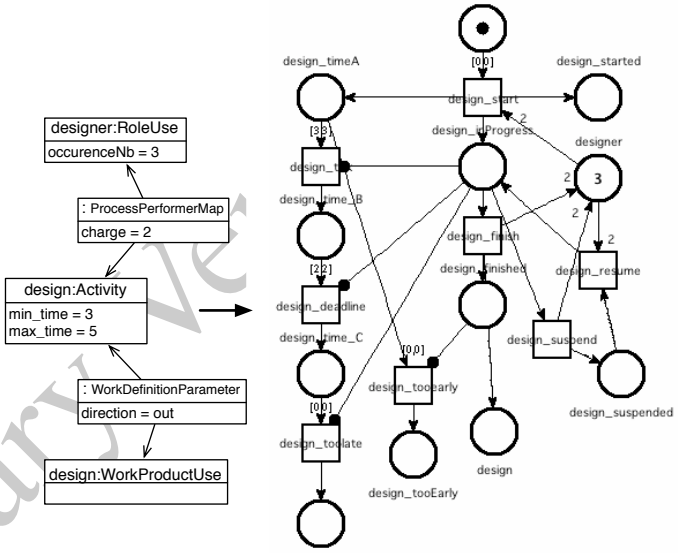

Figure 4. An XSPEM activity as a Petri net

The Petri net model is then translated into the concrete syntax of Tina using an ATL query PETRINET2TINA. To reuse other Petri nets tools, only this last transformation would have do be adapted.

4.2 Model-Checking on Process Models

Now that the process model is translated into a Petri net model, we can check *xSPEM* properties by using Tina. Properties expressed on the xSPEM metamodel leads to an

[2] *TIme Petri Net Analyser*, http://www.laas.fr/tina/

[3] http://eclipse.org/m2m/atl/usecases/SimplePDL2Tina

ATL transformation that produces the corresponding LTL properties instantiated from the xSPEM model.

So, whether a process may be completed or not may becomes a termination problem that can be expressed on the Petri net model. In *xSPEM*, a process is finished when all its activities are finished on time. On the Petri net model, it means that for places of such activities there is one token in the place *_finished* and none in *_tooEarly* and *_tooLate* places. It is defined by the macro-definition `finished` generated from the process model by an ATL query.

We can distinguish the partial correction "all deadlock is in final process state" ($\Box$ (*dead* $\Rightarrow$ `finished`)) and the termination ($\Box \Diamond$ *dead*) which ensures that any execution finishes. At this stage, we have a "strong consistency": any execution finishes and any finished execution is in deadlock. In practice, the constraints expressed in the *xSPEM* model do not allow to finish systematically.

Thus we can be interested by the "weak consistency". It means that at least one execution fulfills the constraints. We evaluate the $- \Diamond$ `finished` property which, literally, expresses that no execution finishes. If the process is weakly consistent, this property is evaluated with *False* and the counterexample produced by `selt` gives one correct execution of the process. If the property is evaluated with *True* then the process does not allow any solution.

Note that the consistency study is generic: it only depends on the `finished` criterium.

5 Project Monitoring

Once process models are validated, an execution support is required in order to ensure orchestrations of activities. At this aim, we can envisage two approaches. The first one consists in developing from scratch, a process engine that will take as input, xSPEM process models in order to execute them. The process engine has then to ensure all enactment facilities such as activity sequencing, resources management, events handling, exceptions processing, etc. The second approach consists in reusing the current state of the art in the Workflow and Business Process Management (BPM) domains. Indeed, these domains have reached a certain maturity level and recently, a consolidation has led to a single language for business process executions: the *Business Process Execution Language for Web Services* (WS-BPEL, BPEL for short) [23]. Rapidly, BPEL gained importance and became the "Language" for business process orchestrations. Many tool vendors already provide training supports and process engines for this standard: ActiveBPEL[4], ApacheAgila[5]. For xSPEM process model executions, we decided to explore the second possibility and

[4]http://www.active-endpoints.com/active-bpel-engine-overview.htm
[5]http://wiki.apache.org/agila/

Table 1. Mappings between xSPEM and BPEL

xSPEM	BPEL
Activity (Outermost)	BPEL Process (next, a BPEL Sequence Activity is required to incorporate nested activities)
Activity (Nested)	BPEL Invoke Activity with a Receive Activity
Activity's properties (min-time and max_time, state, time)	BPEL Variable
WorkProductUse	BPEL Variable
RoleUse	BPEL Variable
WorkSequence (FinishToStart)	BPEL Sequence Activity
WorkSequence (StartToStart, StartToFinish, FinishToFinish)	BPEL Flow Activity combined with the Link element for Synchronization
ProcessParameter	BPEL Variable with attribute MessageType equals to the WorkProductUse used as an Activity ProcessParameter. If the Activity has more than one ProcessParameter than one Part (name=processParameterName) and its type is to be defined for each ProcessParameter whithin the MessageType.
ProcessPerformerMap	BPEL Variable

we opted for BPEL as process execution language. In order to reuse BPEL process engines, a mapping between xSPEM and the process execution language is required. After studying the BPEL standard, in table 1, we propose mappings between xSPEM elements and BPEL concepts.

While establishing these rules we have noticed three major issues. The first one relates to the fact that all xSPEM elements that provide semantics proper to software process modeling have no equivalent in BPEL. All elements such as *RoleUse*, *ProcessPerformerMap*, *WorkProductUse*, etc., are converted into simple BPEL process variables. The second aspect is BPEL's lack of user interactions and support for Human-oriented tasks. Since software processes are mainly composed of human creative tasks, this issue has to be tackled. As a solution, a very interesting work done by industrials known as "BPEL4PEOPLE" [10] can be reused. In BPEL4PEOPLE, a new BPEL activity called *People* activity is introduced. A *People* activity is a basic activity, which is not realized by a piece of software but an action performed by a human being. It can be associated with a group of people, a generic role, etc. The extended BPEL engine creates for each *People* activity a generic user interface in order to highlight inputs/outputs of the activity, deadlines, to add the possibility to attach other materials (e.g., guidelines) and to ease communication between agents. Regarding the implementation, BPEL4PEOPLE leaves the choice to the modeler between five possible configurations. These five configurations, that we will not detail here, fall roughly into two kinds: *Inline Activities* and *Standalone Activities*. Inline activities are defined as part of the BPEL process (they have access to the process context, variables, etc.) while standalone activities are defined outside the process. Standalone activities may be accessed through 1) implementation-specific invocation mechanisms

(i.e., no WSDL), 2) a Web service interface defined with WSDL or 3) a BPEL Invoke activity that calls a Web service implemented by the People activity (WSDL + binding). We opted for the latter configuration. Main reasons are: 1) to promote reusability of standalone activities by other processes, 2) to use tasks in a distributed environment since they offer a WSDL interface, 3) to avoid BPEL engine extensions, since that solution is generic and does not need a support of the new *People* activity kind. However, process modeler can decide to use another configuration among the five that BPEL4PEOPLE proposes if needed. Finally, the last issue relates to the fact that the generated BPEL is not usable straightforward after the transformation. Some data and configuration details have to be set first. Additionally, the process modeler can decide to add new elements or variables for execution aims. This raises the issue of traceability between the XSPEM process model and the generated BPEL process, and how coherence between the two definitions can be preserved.

6 Conclusion

In this paper, we proposed an extension of SPEM2.0 to provide concepts required to enact a process model. A subset of SPEM2.0 is used as a foundation and constitutes a new compliance level. Features have been added to model characteristics of a project, including defining concrete resources allocated to the project and dimensioning activities. We also define features to store process's states during enactment time. Once both process model and the project model have been defined, we propose two complementary approaches. The first one consists in validating models with formal tools (e.g., model-checkers available in the area of Petri nets). It consists in evaluating properties on SPEM2.0 by translating them into LTL properties on the corresponding Petri net model. Properties include termination (i.e., will the process finish) and can also exhibit examples of process planning that fulfill process and project constraints.

The second approach consists in monitoring the project. We proposed a mapping into BPEL, a standard process execution language in the BPM domain. When doing this mapping we identified some drawbacks; the major ones are the loss of semantics proper to software process modeling while mapping SPEM2.0 process models into BPEL, and BPEL's lack of user interaction supports. For this issue, we proposed to reuse the BPEL4PEOPLE proposition. Another deficit of this approach relates to the fact that the code generated after the mapping may need to be completed or modified for execution purposes. However, these modifications are not reflected (traced-up) to the SPEM process model. These approaches are currently evaluated in the context of the IST MODELPLEX and TOPCASED projects. More elaborated prototypes are under construction.

References

[1] R. Bendraou, M.-P. Gervais, and X. Blanc. UML4SPM : A UML2.0-Based Metamodel for Software Process Modelling. In *MoDELS*, volume 3713 of *LNCS*, 2005.

[2] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, July 2004.

[3] S. Chaki, M. E, Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *4th IFM*, volume 2999 of *LNCS*, pages 128–147, Apr. 2004.

[4] T. Clark, A. Evans, P. Sammut, and J. Willans. Applied metamodelling - a foundation for language driven development. version 0.1, 2004.

[5] B. Combemale, P.-L. Garoche, X. Crégut, and X. Thirioux. Towards a Formal Verification of Process Model's Properties – SimplePDL and TOCL case study. In *9th ICEIS*, 2007.

[6] B. Combemale, S. Rougemaille, X. Crégut, F. Migeon, M. Pantel, C. Maurel, and B. Coulette. Towards a Rigorous Metamodeling. In *2nd MDEIS*. INSTICC, May 2006.

[7] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35(9):75–90, 1992.

[8] C. A. Gunter and D. S. Scott. Semantic domains. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 633–674. 1990.

[9] W. Humphrey. Introduction to software process improvement. Technical Report CMU/SEI-92-TR-7, Software Engineering Institute, Carnegie-Mellon University, 1992.

[10] IBM and SAP. *WS-BPEL Extension for People BPEL4People*, July 2005.

[11] F. Jouault and I. Kurtev. Transforming models with ATL. In *MTIP*, volume 3713 of *LNCS*, Jamaica, Oct. 2005.

[12] G. Kahn. Natural semantics. Report 601, INRIA, Feb. 1987.

[13] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving executability into object-oriented meta-languages. In *MoDELS*, volume 3713 of *LNCS*, Montego Bay, Jamaica, Oct. 2005.

[14] Object Management Group, Inc. *Software Process Engineering Metamodel (SPEM) 2.0 RFP*, Nov. 2004.

[15] Object Management Group, Inc. *Software Process Engineering Metamodel (SPEM) 1.1*, Jan. 2005.

[16] Object Management Group, Inc. *Diagram Interchange 1.0 Specification*, Apr. 2006.

[17] Object Management Group, Inc. *Meta Object Facility (MOF) 2.0 Core*, Jan. 2006.

[18] Object Management Group, Inc. *Software Process Engineering Metamodel (SPEM) 2.0*, Mar. 2007.

[19] Object Management Group, Inc. *Unified Modeling Language (UML) 2.1.1 Infrastructure*, Feb. 2007.

[20] Object Management Group, Inc. *Unified Modeling Language (UML) 2.1.1 Superstructure*, Feb. 2007.

[21] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark, 1981.

[22] W. Riddle. Session summary: Opening session. In I. C. Society, editor, *4th International Software Process Workshop (ISPW)*, pages 5–10, Washington, DC, 1989.

[23] WS-BPEL TC OASIS. *Web Services Business Process Execution Language Version 2.0 (Working Draft)*, Jan. 2007.